From Statistical Model Checking to Run-Time Monitoring Using a Bayesian Network Approach

Jaeger, Manfred; Larsen, Kim G; Tibo, Alessandro

Published in:
Runtime Verification - 20th International Conference, RV 2020, Proceedings

DOI (link to publication from Publisher):
10.1007/978-3-030-60508-7_30

Publication date:
2020

Document Version
Accepted author manuscript, peer reviewed version

From Statistical Model Checking to Run-Time Monitoring using a Bayesian Network Approach

Manfred Jaeger[0000−0002−5641−8153], Kim G. Larsen[0000−0002−5953−3384], and Alessandro Tibo[0000−0002−9070−740X]⋆

Aalborg University, Denmark
{jaeger,kgl,alessandro}@aau.dk

Abstract. We propose a framework for monitoring and updating, at run-time, the probabilities of temporal properties of stochastic timed automata. Our method is based on Bayesian networks and can be useful in various real-time applications, such as flight control systems and cardiac pacemakers. The framework has been implemented by exploiting the statistical model checking engine of Uppaal-SMC. By run-time monitoring a set of interesting temporal properties of a given stochastic automaton we update their probabilities, modeled through a Bayesian Network. The main advantages of our method are the capacity to discover non-trivial dependencies between properties and to efficiently update probabilities of unobserved properties given real-time observations. We present empirical results on three application scenarios, showing that the query time can keep up with the speed of some realistic real-time applications. We also present experiments demonstrating that the Bayesian Network approach performance-wise enables run-time monitoring while maintaining or even increasing the accuracy of probability estimation compared to statistical model checking.

Keywords: Timed Automata · Bayesian Networks · Statistical Model Checking.

1 Introduction

Stochastic timed automata are powerful modeling tools for designing and verifying a wide variety of real-time system models, such as real-time monitors for resource management [9], cruise control system [19] in a car, flight control systems [24], and cardiac pacemakers [16, 4]. Precise verification of probabilistic properties for such models quickly becomes intractable, and statistical model checking (SMC)(e.g. [2, 15, 10]) has emerged as a more scalable alternative. However, SMC in many cases will still not be suitable for querying at runtime under real-time constraints, since each probability computation is based on first sampling a number of runs of the stochastic automaton, and the required number of sample runs can become very large when querying probabilities of rare properties. In particular, when we are interested in queries that can be conditioned on

⋆ Authors listed in alphabetical order

partial observations of the system behavior, the SMC approach will often not be viable, since the probability estimation then can only be based on the sampled runs that are consistent with the given observations.

In this paper we therefore develop an alternative, model-based approach. We assume that there is a certain number of key system properties of interest. We then construct a Bayesian network model that represents the joint probability distribution for these selected properties, and that can be efficiently queried for probabilities that are conditioned on complex observations. For example, in a model for a production system involving several processes accessing shared resources, the key properties might be when and for how long which process has accessed which resource, and one could be interested in conditional queries for the expected finishing time of the whole production, given that we have observed the resource usage of some processes up to the present point in time.

We use machine learning algorithms to construct Bayesian network models from the same kind of randomly sampled system runs as used by SMC. As we will see, not only does the Bayesian network model then enable much faster, repeated querying for different conditional probabilities, it also turns out to be more data efficient: a Bayesian network learnt from the same amount of random runs often provides more accurate probability values for the queries of interest, than what is obtained by SMC. This is because the Bayesian network also identifies the (conditional) independence structure between the properties, and thereby can provide accurate probability values even for combinations of properties that never occurred once in the sampled run data.

In this paper we focus on application scenarios where the goal is forecasting properties of system runs based on observations made at runtime. For example, we may want to continuously update a probability estimate for the event that a system's battery reaches a critically low level before the current process on the system terminates. Our method can support other types of applications, however. For example, a Bayesian network model learned from a timed automaton model $\mathcal{M}$ could be used for fault-diagnosis of a real-world implementation $\mathcal{S}$. Internal system states that can be observed in simulations of $\mathcal{M}$ may be unobservable in $\mathcal{S}$, and a model learned from traces of $\mathcal{M}$ can therefore be used to infer hidden (failure) system states of $\mathcal{S}$.

The use of Bayesian networks for diagnostics and forecasting of biological and technical systems is, of course, very well established. However, the usual scenario is that the Bayesian network is either manually constructed by experts, or learned from observational data of the same type of system to which it later will be applied. The novelty of our approach lies in the fact that we base the construction of the Bayesian network on a behavioral model (timed automaton) of the system that we later want to diagnose or monitor, and that a Bayesian network modeling and inference approach then becomes a computational alternative to statistical model checking techniques.

Bayesian networks have previously been used as tools in a runtime verification scenario [17]. However, the nature of the Bayesian networks used in [17], their construction and use is very different from what we propose here. The authors

of [17] use dynamic Bayesian networks (DBNs) which are obtained by a product construction from a system model given as a Hidden Markov Model (HMM), and a property monitor given as a deterministic finite automaton. The nodes of the DBNs then represent time-indexed internal system and monitor states. This is very different from the nodes in our Bayesian networks, which directly represent properties of interest of entire system traces. Whereas the DBNs of [17] are obtained by a deterministic constructions, ours are learned from simulation data. Finally, the objective of [17] is online monitoring of the current system state. As explained above, our focus is on forecasting future events and properties in the full system run.

We implemented and evaluated our approach based on Uppaal-SMC [10] as the modeling platform for stochastic timed automata, as the sample generator for random system runs, and as the SMC tool that we compare against.

The paper is organized as follows. In Section 2 we formally review some background concepts related to timed automata, statistical model checking, and Bayesian Networks. In Section 3 we present the proposed framework. In Section 4 we report an experimental evaluation of our method on a real-case scenario, introduced at a first as toy example for a better explanation and then two concreted cases. We also perform a comparison between statistical model checkers and Bayesian Networks in Section 5. Finally, in Section 6 we draw some conclusions and discuss possible extensions as future work. The code we used for the experiments can be downloaded from https://github.com/alessandro-t/uppaal-bn.

2 Background

In this section we will review the ingredients of our framework: timed automata models, statistical model checking, and Bayesian Networks.

2.1 Timed Automata

Timed Automata are finite automata enriched with real-valued variables called clocks [3]. Clocks measure the progress of time which elapses while an automaton is residing in some location. Transitions between locations can be constrained based on clock values and clocks may be reset on transitions. In the tool Uppaal [20] the modelling formalism is extended to networks of timed automata, including handshake and broadcast communication primitives, communication over shared variables as well as a C-like imperative programming language which allows transitions to be conditioned on and perform complex updates of discrete structured variables.

Consider the small safety critical Bridge Scenario depicted in Figure 1. Here a `Car` needs to cross over bridge, while at the same time a `Ship` wants to pass under the bridge. However, for the `Ship` to make the passing safely the bridge needs to be open. Conversely, the bridge needs to be closed in order for the `Car` to safely cross over the bridge. Obviously, the bridge is only allowed to be opened in case the `Car` is not on the bridge.

Figure 2 provides a timed automata based model of the Bridge Scenario in UPPAAL. The model consists of two timed automata `Car` and `Ship` with their respective clocks `x` and `y` used to constrain the timing of their transitions: e.g. the combination of the invariant `x<=10` of the location `Car.Init` and the guard `x<=5` implies that the output action `carW!` will take place after a delay of d time-units with $5 \leq d \leq 10$. When reaching the various locations, both `Car` and `Ship` broadcast relevant output actions, `Car` broadcasts `carB!` when entering the bridge-location `Car.B`. The two timed automata synchronize exclusively using the two boolean variables `Bcl` (indicating that the bridge is closed) and `Bus` (indicating that the `Car` is on the bridge). Semantically, a timed automaton describes a timed labelled transition systems, with states being pairs (ℓ, ν), where ℓ is a location and ν is a valuation for the clocks, and with transitions being either delays or discrete actions. In a network of extended timed automata, states are vectors of states – one per component – together with concrete values of discrete variables. E.g. in the Bridge Scenario model of Figure 2 $[(\texttt{Car.B}, \texttt{x} = 0), (\texttt{Ship.W}, \texttt{y} = 11.9), \texttt{Bus} = 1, \texttt{Bcl} = 1]$ is a reachable state, witnessed by the following transition sequence:

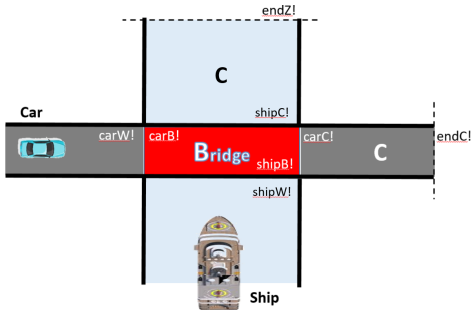

Fig. 1. Safety Critical Bridge Scenario involving a `Car` and a `Ship`.

$$
\begin{aligned}
&\big[(\texttt{Car.Init}, \texttt{x} = 0), (\texttt{Ship.Init}, \texttt{y} = 0), \texttt{Bus} = 0, \texttt{Bcl} = 1\big] \\
&\xrightarrow{7.28}\xrightarrow{\texttt{carW!}} \big[(\texttt{Car.W}, \texttt{x} = 0), (\texttt{Ship.W}, \texttt{y} = 7.28), \texttt{Bus} = 0, \texttt{Bcl} = 1\big] \\
&\xrightarrow{4.62}\xrightarrow{\texttt{carB!}} \big[(\texttt{Car.B}, \texttt{x} = 0), (\texttt{Ship.W}, \texttt{y} = 11.9), \texttt{Bus} = 1, \texttt{Bcl} = 1\big].
\end{aligned}
$$

Given the timed automata model of the Bridge Scenario in Figure 2, the symbolic verification engine of UPPAAL allows us to verify the crucial safety property `A[] !(Car.B and Ship.B)`, which is a CTL formula expressing that the `Ship`

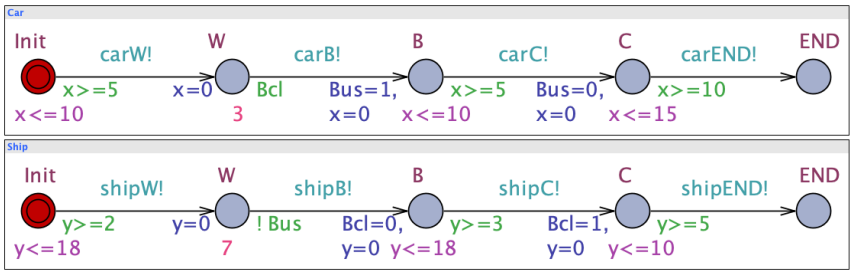

Fig. 2. Timed automata model of the Bridge Scenario of Figure 1.

and the `Car` cannot be under/at the bridge at the same time. In addition, we may be interested in knowing whether the `Car` or the `Ship` may reach their respective `END`-location first. In fact, both outcomes are possible as may be witness by model checking the reachability properties `E<> !Car.END and Ship.END` and `E<> Car.END and !Ship.END`.

2.2 Statistical Model Checking

Beyond crucial safety properties, we are often interested in more refined performance analysis of a system. E.g. for the Bridge Scenario, we would be interested in the probabilities that the `Car` or the `Ship` finishes first. For such performance queries to be meaningful, we need a *stochastic* semantics of networks of timed automata, where the non-deterministic choices of delays are refined by probability distributions, and where non-deterministic choices between discrete actions are refined by probabilistic choices. In the branch Uppaal-SMC, delays of components are by default resolved by a uniform distribution (e.g. in Figure 2 the delay of `Car` in `Init` is uniform between 5 and 10 time-units) or an exponential distribution (e.g. in Figure 2 the delay of `Ship` in `W` is given by an exponential distribution with rate 7). Other distributions may be specified by the user. For composite systems, the choice of which component will perform the next output action (and at which time) is a race between the components settled stochastically by the independent delay distributions of these. We refer the reader to [12, 13, 11] for more details on the semantics of stochastic timed automata adopted in Uppaal-SMC.

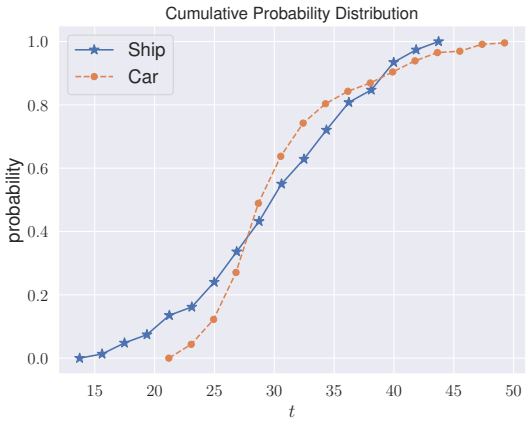

Fig. 3. Cumulative Distributions obtained by `Pr[<=T](<> Ship.END)` and `Pr[<=T](<> Car.END)`

Crucially, the stochastic semantics of timed automata offers the basis of a probability measure on measurable sets of runs (obtained from a natural cylinder-construction). In fact, time-bounded reachability, safety properties as well as Metric Interval Temporal Logic (MITL) properties all describe measurable sets of runs [6]. Based on Monte Carlo simulation, the engine of Uppaal-SMC allows to estimate the probabilities of such (time-bounded) properties by confidence intervals (the user-desired size and confidence of which determines the number of simulations needed). In Figure 3, we see that the most likely "winner" of the `Car` and `Ship` depends highly on the timing bound.

Rather than using MITL or (other logics) for expressing properties of runs, we will use *monitors*, being purely inputting, deterministic timed automata, that are added as extra parallel components to the system. Given a set of (absorbing) accept locations, a monitor M describes all the timed words ϕ_M over the alphabet of the system, that will lead to an accept location. Now, the engine of UPPAAL-SMC allows to estimate $p(\phi_M)$, i.e. the probability that a random run of the system will be accepted by M. In [14] monitors are used to express properties of continuous-time Markov chains. In [1] the logical power of timed automata monitors is characterized.

Monitors M express *logical* properties ϕ_M of runs, i.e. for any given run π, $\phi_M(\pi)$ is either true or false. In this paper, we consider the generalization to *categorical* properties ψ, which for any run π returns a value $\psi(\pi)$ from a finite domain $V = \{v_1, \ldots, v_n\}$. A *categorical monitor* C over V is a monitor, but with n designated terminal and absorbing states $S_a = \{s_1, \ldots s_n\}$ rather than a set of accept states. Now the categorical property ψ_C realized by C is simply $\psi_C(\pi) = v_j$ whenever the run π reaches the terminal state s_j. For the purpose of this paper, we shall assume that monitors reach an absorbing state with probability one. In fact, for the models we use in our examples, the stronger property holds that there exists a fixed upper time bound T, such that monitors will reach an absorbing state after a most T time-units of the underlying system.

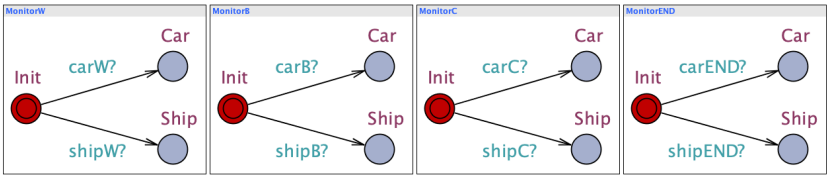

Fig. 4. Monitors from left to right, representing the properties $\phi_{\mathtt{W}}$, $\phi_{\mathtt{B}}$, $\phi_{\mathtt{C}}$, and $\phi_{\mathtt{END}}$.

For our bridge example, we can consider for $\mathtt{S} \in \{\mathtt{W}, \mathtt{B}, \mathtt{C}, \mathtt{END}\}$ the categorical properties $\phi_{\mathtt{S}}$, all with domain $V = \{\mathtt{Car}, \mathtt{Ship}\}$, and defined such that $\phi_{\mathtt{S}}(\pi) = \mathtt{Car}$ iff in the run π car is first to reach $\mathtt{S}$. Figure 4 shows the monitors for these four properties.

From general stochastic timed automata and the very powerful specification language of monitors, we obtain probabilistic queries $P(\psi_C(\pi) = v_j) =?$ that are undecidable, and therefore outside the reach of exact probabilistic model checking [5]. Approximate estimation techniques like SMC, or the Bayesian network model-based approach we introduce here, therefore are required.

2.3 Bayesian Networks

A (categorical) Bayesian Network is a directed acyclic graph $G = (V, E)$, where each node $v_i \in V$ is associated with a categorical random variables X_i, and

edges represent dependencies between the random variables [18]. Let $N := |V|$. G defines the joint probability of the random variables as a product of conditional distributions:

$$p(X_1, \ldots, X_N) := \prod_{i=1}^{N} p(X_i | Parents(X_i)),$$

where $Parents(X_i) = \{X_j | j : (v_j, v_i) \in E\}$. In our application, we need to solve the following computational problems:

Structure learning. Given a dataset of K observations $(x_1^i, \ldots, x_N^i)$ $(i = 1, \ldots, K)$ of the random variables, learn the edges E of G by optimizing an objective that combines the maximum likelihood criterion with a simplicity objective (sparser structures are preferred). Out of several objective functions that embody similar principles, we choose the Bayesian Information Criterion [22] (BIC) score. The BIC score is asymptotically consistent in the sense that in the large sample limit the BIC-optimal structure will be a minimal structure (in terms of the number of numerical parameters it contains) over which the data generating distribution can be represented [8]. Identifying the BIC-optimal Bayesian network structure is in general NP-hard in the number of variables [7]. On the other hand, it can be shown that, again in the large sample limit, greedy search strategies are sufficient to identify the optimal structure [8]. However, these optimal greedy strategies operate in complex and highly connected search spaces, so that even though the number of search steps they require is polynomial, their overall complexity is not. In all our experiments, we maximize BIC score via greedy Hill Climbing [21], as implemented in the Python library bnlearn [23] for structure learning. We note that even though hill climbing does not necessarily give us the BIC-optimal structure, it will give us in the large sample limit an over-approximation, i.e., a structure that can represent the data-generating distribution, even though not necessarily the sparsest possible such structure. As in our context we are able to generate arbitrary amounts of training data, we can, in principle, approximate the true distribution with arbitrary precision.

Parameter Learning. Once the structure of G is learnt, the parameters of the conditional distributions $p(X_i | Parents(X_i))$ are estimated by maximizing the likelihood. In our case of categorical random variables, and the availability of complete data (the values of all random variables X_i are observed in all of the K samples), this amounts to nothing more than calculating relative frequencies of the value configurations that are needed for a tabular representation of the conditional distributions.

Inference. The learnt Bayesian network is used to compute conditional probability distributions $p(X_i | X_{h_1} = x_{h_1}, \ldots, X_{h_m} = x_{h_m})$ of a query variable X_i given observed values for a subset of the remaining variables. In the worst case, this probabilistic inference is still NP-hard in the size of the Bayesian network. However, several inference techniques exist that often lead to efficient inference in practice. We make use of the Variable Elimination [25] (VE) algorithm, as implemented in the python library bnlearn [23].

3 Methodology

We now describe in detail our approach to combine the power of stochastic timed automata for modeling complex systems with the ability of Bayesian networks for fast and flexible computations of conditional probability distributions. The fundamental assumption of our approach is that we have at our disposal a stochastic timed automaton, and that we have identified a number of relevant categorical path properties $\phi_1, \ldots, \phi_N$ represented by monitors, such that we need to compute conditional probability distributions involving these properties either over-and-over again, or under real time constraints at run time (or both). In the first case our approach will have an advantage over SMC in terms of amortized time complexity (the time needed to construct the Bayesian network will be more than compensated by much faster computations of query probabilities). In the second case SMC may not be feasible at all because of its inability to meet the time constraints.

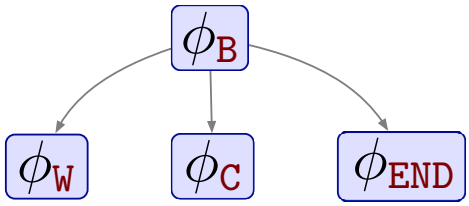

Fig. 5. The Bayesian Network learnt from data generated by the model of Figure 2 and monitors of Figure 4.

We collect data by running K simulations of the stochastic timed automaton, and recording for each simulation the values returned by the N monitors. This data is collected in a data table D of dimensions $K \times N$, where the ith column then contains values from the domain of ϕ_i.

From D we learn both the structure and the parameters for a Bayesian Networks G, as described in Section 2. Figure 5 shows a Bayesian network for the path properties defined by the monitors of Figure 4 that was learnt from $10,000$ random executions of the model shown in Figure 2. The model learning has identified the $\phi_{\mathtt{B}}$ property as the central random variable, and correctly shows that given the information which of `Car` or `Ship` first passes the bridge, the random variables $\phi_{\mathtt{C}}$ and $\phi_{\mathtt{END}}$ become independent of $\phi_{\mathtt{W}}$.

G can be now used to query the probability distribution of any of the properties, given arbitrary observations of other properties. We illustrate two prototypical query scenarios. The first one is *runtime forecasting*: here we want to continuously update the prediction of a variable as the system run evolves. Figure 6 shows an example where the query random variable is $\phi_{\mathtt{END}}$, and we update the probability distribution for $\phi_{\mathtt{END}}$ each time the value of one of the other variables becomes known. Figure 7 shows instead the explicit updates of

Table 1. $p(\phi_{\texttt{W}} = \texttt{Car})$ given the four possible combinations for $\phi_{\texttt{C}}$ and $\phi_{\texttt{END}}$.

	$\phi_{\texttt{END}} = \texttt{Car}$	$\phi_{\texttt{END}} = \texttt{Ship}$
$\phi_{\texttt{C}} = \texttt{Car}$	0.961	0.5
$\phi_{\texttt{C}} = \texttt{Ship}$	0.005	0.005

the Bayesian network of Figure 5 as new properties are observed. We observe that during this particular run of the system the believed probability of the Car being the first at END decreases radically at time-point $t = 10.53$. The second scenario is *diagnostic prediction* for unobservable properties. Even though all the properties in the Bayesian network correspond to monitors in the timed automata model, in the real world some of these properties may not be observable. We can then use the Bayesian network to make predictions about the unobservable properties based on what we could observe. In our example, we may assume that only $\phi_{\texttt{C}}$ and $\phi_{\texttt{END}}$ are observable, due to suitably positioned cameras. We may then be interested in inferring the state of $\phi_{\texttt{W}}$ given the available observations. $\phi_{\texttt{W}}$ is an interesting property, for example, if one aims to minimize the waiting time of the boat as this latter pollutes more than a car. Table 1 shows the probability values for $\phi_{\texttt{W}} = \texttt{Car}$ given the four possible configurations of the observable variables.

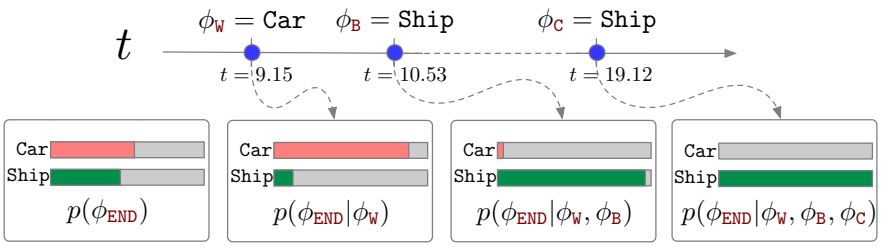

Fig. 6. Evolution of $p(\phi_{\texttt{END}})$ as new properties are observed. Note that right after $t = 9.15$ the Car most likely arrives first to the end. However, at $t = 10.53$ the situation drastically changes due to the observation of $\phi_{\texttt{B}} = \texttt{Ship}$.

4 Experiments

We evaluated our method on three different scenarios.

4.1 Bridge Crossing

We consider a more complex set of properties for the Bridge scenario described in Section 1. We now define properties $\psi_{\texttt{Car},\texttt{S}}$, $\psi_{\texttt{Ship},\texttt{S}}$ for $\texttt{S} \in \{\texttt{W}, \texttt{B}, \texttt{C}, \texttt{END}\}$.

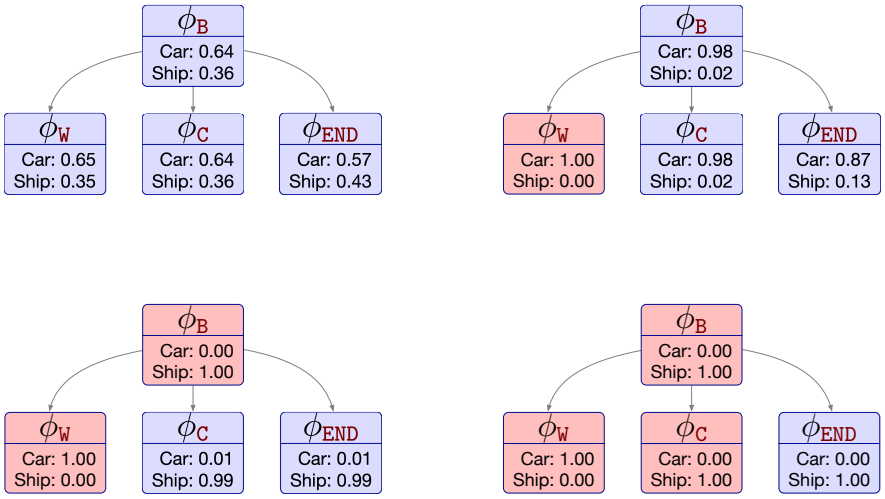

Fig. 7. The explict Bayesian network updates as new properties are observed. The red block represents properties observations.

Associated with these properties are domains $T_{\mathtt{Car},\mathtt{S}}, T_{\mathtt{Ship},\mathtt{S}}$ that each consist of 8 time intervals.

For example, $T_{\mathtt{Car},\mathtt{B}} = \{[0, 5.931], [5.931, 6.601], \ldots, [19.195, 28.614]\}$, and $\psi_{\mathtt{Car},\mathtt{B}} = [19.195, 28.614]$ means that the car has arrived at the bridge between time points 19.195 and 28.614. To define the time interval domains for each variable we used a random sample of system runs, and quantized the empirically observed arrival times into 8 bins so that all bins contain an equal numbers of sample points. As a consequence, prior to any observations, the probability distribution for each ψ variable is uniform over its domain (cf. Fig. 9 at $T = 0$). We also retain the property $\phi_{\mathtt{END}}$ with domain $\{\mathtt{Car}, \mathtt{Ship}\}$ as introduced in Section 3.

We here consider a runtime forecasting scenario, where we want to maintain a continuously updated prediction for the time $\psi_{\mathtt{Car},\mathtt{END}}$ at which the car reaches its destination. We learnt a Bayesian network for the joint probability distribution of the properties from a dataset of $K = 10,000$ model simulations. The resulting Bayesian Network is depicted in Figure 8. Figure 9 reports for one particular run the evolution of $p(\psi_{\mathtt{Car},\mathtt{END}})$ as new properties are observed, and hence used as evidence. Here, new evidence is obtained at 4 distinct points in time $T_1, \ldots, T_4$. The figure shows for each of these timepoints the newly acquired observation, and the conditional probability distribution of $p(\psi_{\mathtt{Car},\mathtt{END}})$ given all observations up to that point in time.

Finally, we estimated the time per query by averaging the query time on $10,000$ model traces, which results to be 0.08 ± 0.05 seconds. Those results confirm that the Bayesian Network query can keep up with the speed of this real-time application.

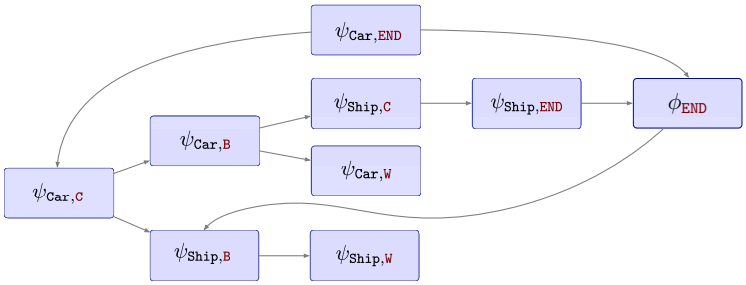

Fig. 8. The Bayesian Network structure for the Bridge Scenario.

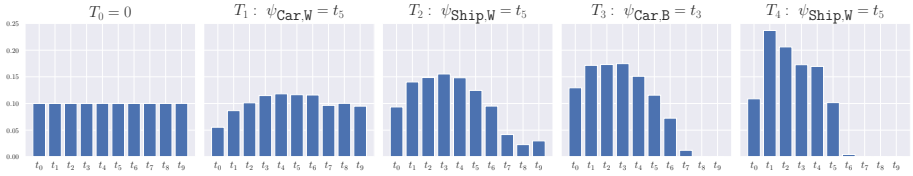

Fig. 9. The evolution of $p(\psi_{\texttt{Car,END}})$ as new properties are observed:

4.2 Job Shop

Three persons `Kim`, `Manfred`, and `Alessandro`, want to read the four sections `com`, `spo`, `pol`, and `loc` (*Commerce*, *Sport*, *Politics*, and *Local news*) of a shared newspaper. Each person can read one section at a time and the sections cannot be shared among the persons. A person must wait until a section becomes available before reading it. Furthermore, each person reads the sections in a given order for a different amount of time. Figure 10 depicts an example of ordered section requests for each person. For each person, we designed a template in UPPAAL-SMC, an example of which is shown in Figure 11. During each simulation of the model we are interested in several temporal properties. Let $U = \{\texttt{Kim}, \texttt{Manfred}, \texttt{Alessandro}\}$ be the set of persons, $S = \{\texttt{com}, \texttt{spo}, \texttt{pol}, \texttt{loc}\}$ the set of sections, and $T_U = \{t_0, \ldots, t_W\}$ and $T_S = \{t_0, \ldots, t_Z\}$ sets of time intervals. We are interested in evaluating the following properties during each simulation of the model

- for each $u \in U$ the property ϕ_u with domain T_U representing the time interval in which u finishes the reading or all sections.
- for each $u \in U$ and $s \in S$ the property $\psi_{u,s}$ with domain T_S representing the time interval in which u finishes to read section s.

We again learnt a Bayesian network for the joint probability distribution of the properties from a dataset of $K = 10,000$ model simulations. The resulting Bayesian Network is depicted in Figure 12. For this case, the learnt structure shows two main clusters which involves `Alessandro` and `Kim` properties. The cluster related to `Alessandro` connects more specific `Alessandro` properties. A

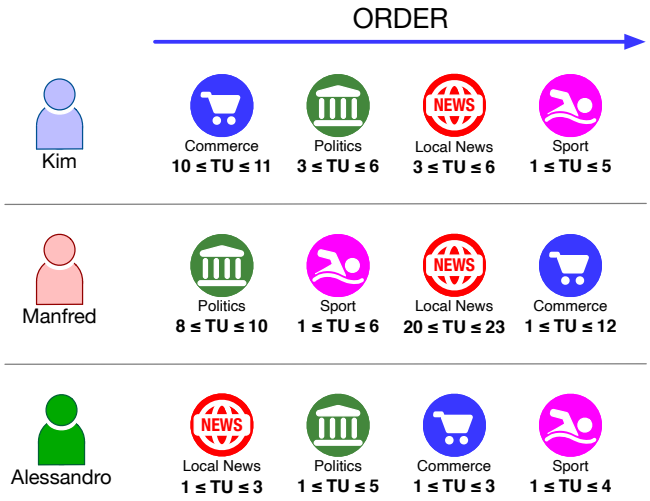

Fig. 10. Example of ordered section requests for `Kim`, `Manfred`, and `Alessandro`. TU represents the time units for which each section is requested by a person, e.g. `Kim` asks for the `com` section for at least 10 TU and for at most 11 TU.

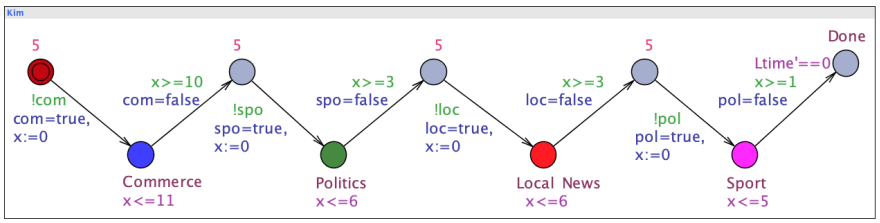

Fig. 11. UPPAAL-SMC template for `Kim`. The green statements represent conditions for executing a transition. The blue statements represent variable updates right after a transition is executed. The purple statements represents invariants. `com`, `pol`, `loc`, and `spo` are boolean variables representing whether the corresponding section (*Commerce*, *Sport*, *Politics*, and *Local news*) is free. `x` is a clock.

similar behaviour happens for `Kim`, while the properties related to `Manfred` are more distributed over the network.

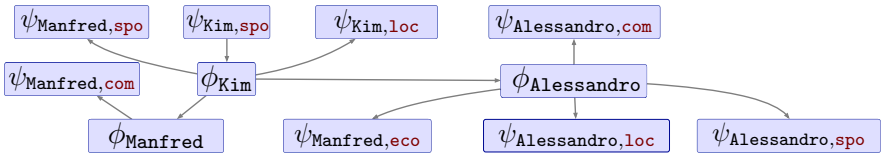

Fig. 12. The Bayesian Network structure for the Job Shop Scenario. Four of the properties are represented by nodes not connected to any other nodes. For the sake of better visualization they are not shown in the figure.

With the trained Bayesian Network we can now infer (on new simulations) probabilities of unobserved properties given the knowledge of real-time observed properties. Figure 13 shows an example of the evolution of $p(\phi_{\texttt{Kim}})$ as new properties are observed and hence used as evidence. Finally, we report in Figure

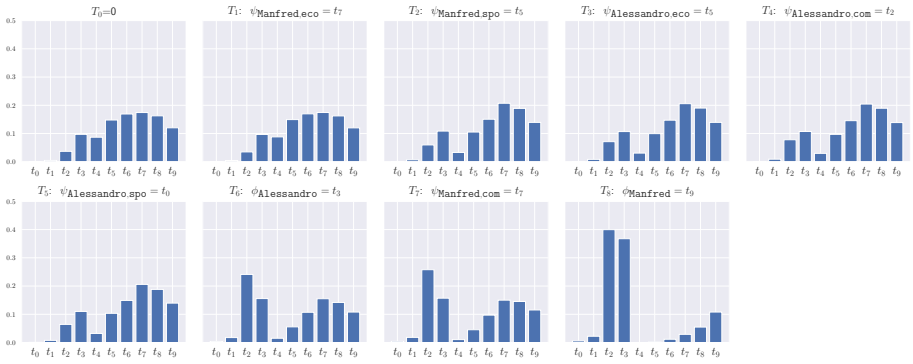

Fig. 13. The evolution of $p(\phi_{\texttt{Kim}})$ as new properties are observed.

14 the empirical query time distribution on 10,000 model traces, similarly to Section 4.1. We point out the fact that the estimated query time distribution has small variance and is concentrated around 0.04 seconds.

4.3 Process Resource Model

In this section we introduce a general process resource model, which can be seen as a common abstraction of the two previous examples. Here, we assume that we monitor a set of processes which, for being able to complete their execution, have to access a set of resources in a certain order. Each resource can be accessed

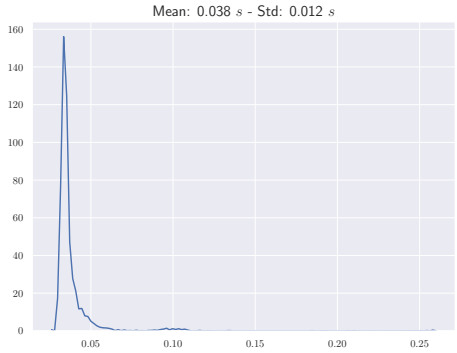

Fig. 14. Empirical time distribution estimated on 10,000 traces.

only by one process at a time and each process can use only one resource at a time. Furthermore, each process cannot reuse the same resource.

We are here interested in studying the scalability of our approach by measuring the execution time for the queries, when the complexity of the system, as given by the number of processes and resources, increases. We consider three different instantiation of the generic system model:

- `System A`: 5 processes and 5 resources;
- `System B`: 10 processes and 10 resources;
- `System C`: 20 processes and 20 resources.

For this scenario we consider two sets of properties: one that contains properties which capture the ending time for each process, and one that contains properties which capture the duration for which each process uses a certain resource. Each property has a domain of 5 time intervals. For a system with n processes and n resources, this gives us a total of $C(n) := n + n^2$ properties. Similarly to the previous experimental sections, for each of the three systems we trained a Bayesian Network on data from 5,000 sample runs. The learnt models for `A`, `B`, `C` have $C(n)$ nodes, and 14,16 and 22 edges, respectively. This very sparse connectivity is due to the fact that the properties are mostly very weakly dependent, and the BIC score used for training will approximate sufficiently weak dependence by independence assumptions. The computation times for learning the structure of the three Bayesian Networks are $38s$, $373s$, and $1914s$ (averaged over 5 different learning runs).[1]

We simulated a runtime forecasting scenario as follows: for each system we generated another 5,000 random runs. For each run, we selected the property whose value was observed last on that run (i.e., the finishing time of the process that finished last on the run) as the query property. Then, similar to what is

[1] All the experiments ran on an Apple MacBook Pro Mid 2015, 2.5 GHz Quad-Core Intel Core i7, with 16 GB of RAM and only one core has been used.

shown in Figures 9 and 13, we computed over each run the sequence of conditional probability distributions of the query property, given the incremental observation sequence. Thus, for a system of size n, a total of $5,000 \cdot C(n)$ probability queries conditioned on 0 to $C(n) - 1$ observed properties were computed.

Figure 15 shows the resulting empirical query time distribution for `System A`, `System B`, and `System C`. The results show that here our approach scales very well as the size of the system increases. The almost constant query time here is enabled by the weak dependencies among the properties, and the resulting sparse connectivity of the Bayesian network. Such an independence structure cannot be exploited in a similar manner by SMC, as our following results show.

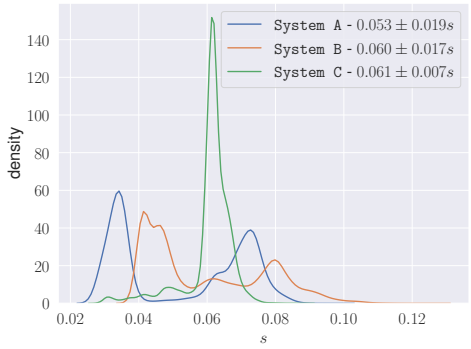

Fig. 15. Distribution of the query time execution for `System A`, `System B`, and `System C`.

We ran the same queries that give the results of Figure 15 in Uppaal-SMC. Since these are conditional probability queries which are not directly supported by SMC, we calculate a conditional probability $p(A|B)$ as the ratio $p(A,B)/p(B)$, using two separate SMC estimates for $p(A,B)$ and $p(B)$. Uppaal-SMC requires the specification of a confidence parameter ϵ for the width of a confidence interval that will be returned for the query probability. When $[l_{A,B}, u_{A,B}]$ and $[l_B, u_B]$ are the confidence intervals for $p(A,B)$ and $p(B)$, respectively, we compute the confidence interval for $p(A|B)$ as $[\frac{l_{A,B}}{u_B}, \frac{u_{A,B}}{l_B}]$. This can lead to very wide confidence intervals for the conditional, even when $[l_{A,B}, u_{A,B}]$ and $[l_B, u_B]$ have a small width ϵ. Ideally, we would use for this experiment ϵ values such that the confidence intervals for the conditional probabilities become reasonably small, and the accuracy of the SMC estimate matches the accuracy of the Bayesian network values. This is not an easy task, and as the results of the next section will show, would probably require ϵ values that are much smaller than what can feasibly be computed. For the purpose of this runtime comparison we therefore simply consider the three values $\epsilon = 0.1, 0.01, 0.001$, which will only give very rough estimates for conditionals once the probabilities of the conditioning observations become small.

Table 2 shows the average query time for the three models and three ϵ values. We set a timeout at 5 minutes. We observe that SMC computation here scales neither as a function of the system complexity, nor as a function of the precision parameter ϵ.

Table 2. Average time per query in UPPAAL-SMC

ϵ	SYSTEM A	SYSTEM B	SYSTEM C
0.1	$0.13s$	$0.52s$	$3.68s$
0.01	$1.60s$	$8.22s$	$> 5m$
0.001	$> 5m$	$> 5m$	$> 5m$

5 Accuracy Analysis

In this section we investigate the accuracy of probabilities calculated with learnt Bayesian networks, and compare against the accuracy that is obtained by SMC using a comparable amount of data.

We return to the Bridge scenario described in Sections 1 and 4.1. We consider the conditional probability distribution

$$p(\phi_{\texttt{END}} = \texttt{Car} | \psi_{\texttt{Car,W}}, \psi_{\texttt{Ship,W}}, \psi_{\texttt{Car,B}}, \psi_{\texttt{Ship,B}}).$$

For the conditioning variables $\psi_{\texttt{Car,W}}, \ldots, \psi_{\texttt{Ship,B}}$ we identify a *rare* and a *common* joint configuration of time-interval values. The rare and common configuration occur 1177, respectively 6147 times in a sample of $1M$ traces.

We estimated the true query probabilities by repeating 10 simulations of $1M$ traces each. The mean empirical values of the query probabilities, and the standard deviation over the 10 simulations is shown in Figure 16 as "Real". The negligible standard deviation shows that we can treat the obtained estimate as the ground truth probability.

We learnt Bayesian networks from K sampled traces for 14 distinct values of K ranging from $K = 50$ to $K = 600,000$. The probabilities obtained from the learnt networks for the two different queries are plotted in Figure 16 as a function of K. As described in Section 4.3, SMC in UPPAAL-SMC is not directly controlled by specifying a sample size, but through a confidence parameter ϵ. We ran UPPAAL-SMC multiple times, continuously varying ϵ from 0.05 to 0.00005, and computing a confidence interval for the conditional query from the confidence intervals at the specified ϵ level for the two unconditional probabilities. In Figure 16 the resulting confidence interval is plotted against the size of the sample that was actually required to obtain the confidence intervals at the specified ϵ. The plot also shows the mid-point of the obtained confidence intervals as point estimates for the query probabilities.

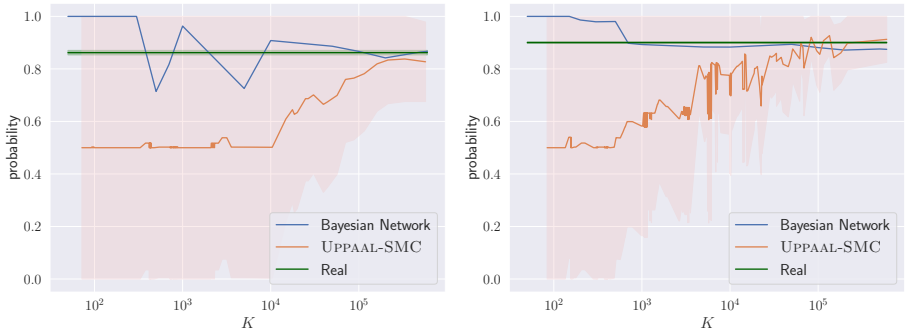

Fig. 16. $p(\phi_{\texttt{END}} = \texttt{Car} | \psi_{\texttt{Car,W}}, \psi_{\texttt{Ship,W}}, \psi_{\texttt{Car,B}}, \psi_{\texttt{Ship,B}})$ estimated with Bayesian Network in blue and Uppaal-SMC in orange. In green the true probability. Left/right: estimated probabilities for the rare/common configuration of observed properties.

There are a number of observations we can make from these plots: first, we see that even for very small ϵ values (i.e., large sample sizes), the width of the SMC confidence interval for the conditional probability decreases only very slowly. This, in particular, indicates that the ϵ values shown in Table 2 are still larger than what would be required to obtain reasonably accurate probability estimates. The point estimates obtained from the Bayesian network converge to the true probability much faster than the SMC point estimates. This is more pronounced in the 'rare' configuration, where SMC is handicapped by the smaller number of samples that will be relevant for the query probability. The Bayesian network estimates are less affected by the rarity of the observed values, since its probabilities are derived from a combination of empirical frequencies, and inferred conditional independence relationships.

6 Conclusions

We have introduced a framework which links statistical model checking with machine learning. We exploited Bayesian Networks to learn dependencies among interesting temporal properties of a stochastic timed automaton. We have identified real-time application scenarios where our approach can be used. In particular, we highlight the advantages of the Bayesian Network approach in comparison with SMC for real-time updating of probabilities of unobserved properties as new property are observed during the evolution of a real-time system. We empirically validated our framework on three real-time scenarios, showing that Bayesian Network inference is able to keep up with the evolution of a real-time system. Furthermore, the estimated probabilities are at least as accurate as what is obtained from Uppaal-SMC.

It is a interesting subject of future work to extend our approach to learn also a set of interesting properties, rather than defining the properties a priori.

This research direction might lead to strengthen the connections between the machine learning and the statistical model checking domains.

References

1. Aceto, L., Bouyer, P., Burgueño, A., Larsen, K.G.: The power of reachability testing for timed automata. In: Arvind, V., Ramanujam, R. (eds.) Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1530, pp. 245–256. Springer (1998). https://doi.org/10.1007/978-3-540-49382-2_22, https://doi.org/10.1007/978-3-540-49382-2_22
2. AlTurki, M., Meseguer, J.: Pvesta: A parallel statistical model checking and quantitative analysis tool. In: International Conference on Algebra and Coalgebra in Computer Science. pp. 386–392. Springer (2011)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8, https://doi.org/10.1016/0304-3975(94)90010-8
4. Alur, R., Giacobbe, M., Henzinger, T.A., Larsen, K.G., Mikučionis, M.: Continuous-time models for system design and analysis. In: Computing and Software Science, pp. 452–477. Springer (2019)
5. Bulychev, P., David, A., Larsen, K.G., Mikučionis, M., Poulsen, D.B., Legay, A., Wang, Z.: Uppaal-smc: Statistical model checking for priced timed automata. arXiv preprint arXiv:1207.1272 (2012)
6. Bulychev, P.E., David, A., Larsen, K.G., Legay, A., Li, G., Poulsen, D.B.: Rewrite-based statistical model checking of WMTL. In: Qadeer, S., Tasiran, S. (eds.) Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7687, pp. 260–275. Springer (2012). https://doi.org/10.1007/978-3-642-35632-2_25, https://doi.org/10.1007/978-3-642-35632-2_25
7. Chickering, D.M., Heckerman, D., Meek, C.: Large-sample learning of bayesian networks is np-hard. Journal of Machine Learning Research **5**(Oct), 1287–1330 (2004)
8. Chickering, D.M., Meek, C.: Finding optimal bayesian networks. In: Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence. pp. 94–102 (2002)
9. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal stratego. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 206–211. Springer (2015)
10. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal smc tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397–415 (2015)
11. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transf. **17**(4), 397–415 (2015). https://doi.org/10.1007/s10009-014-0361-y, https://doi.org/10.1007/s10009-014-0361-y
12. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (eds.) Formal Modeling and Analysis of

Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6919, pp. 80–96. Springer (2011). https://doi.org/10.1007/978-3-642-24310-3_7, https://doi.org/10.1007/978-3-642-24310-3_7
13. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 349–355. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_27, https://doi.org/10.1007/978-3-642-22110-1_27
14. Feng, Y., Katoen, J., Li, H., Xia, B., Zhan, N.: Monitoring ctmcs by multi-clock timed automata. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 507–526. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_27, https://doi.org/10.1007/978-3-319-96145-3_27
15. Jégourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking - PLASMA. In: Flanagan, C., König, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7214, pp. 498–503. Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_37, https://doi.org/10.1007/978-3-642-28756-5_37
16. Jiang, Z., Pajic, M., Moarref, S., Alur, R., Mangharam, R.: Modeling and verification of a dual chamber implantable pacemaker. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 188–203. Springer (2012)
17. Kalajdzic, K., Bartocci, E., Smolka, S.A., Stoller, S.D., Grosu, R.: Runtime verification with particle filtering. In: International Conference on Runtime Verification. pp. 149–166. Springer (2013)
18. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
19. Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Safe and optimal adaptive cruise control. In: Correct System Design, pp. 260–277. Springer (2015)
20. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1-2), 134–152 (1997). https://doi.org/10.1007/s100090050010, https://doi.org/10.1007/s100090050010
21. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall Press, USA, 3rd edn. (2009)
22. Schwarz, G., et al.: Estimating the dimension of a model. The annals of statistics **6**(2), 461–464 (1978)
23. Taskesen, E.: bnlearn. https://github.com/erdogant/bnlearn (2019)
24. Wu, X., Ling, H., Dong, Y.: On modeling and verifying of application protocols of ttcan in flight-control system with uppaal. In: 2009 International Conference on Embedded Software and Systems. pp. 572–577. IEEE (2009)
25. Zhang, N.L., Poole, D.: A simple approach to bayesian network computations. In: Proceedings of the biennial conference-Canadian society for computational studies of intelligence. pp. 171–178. CANADIAN INFORMATION PROCESSING SOCIETY (1994)